Parveen Khurana

124 Followers About Following

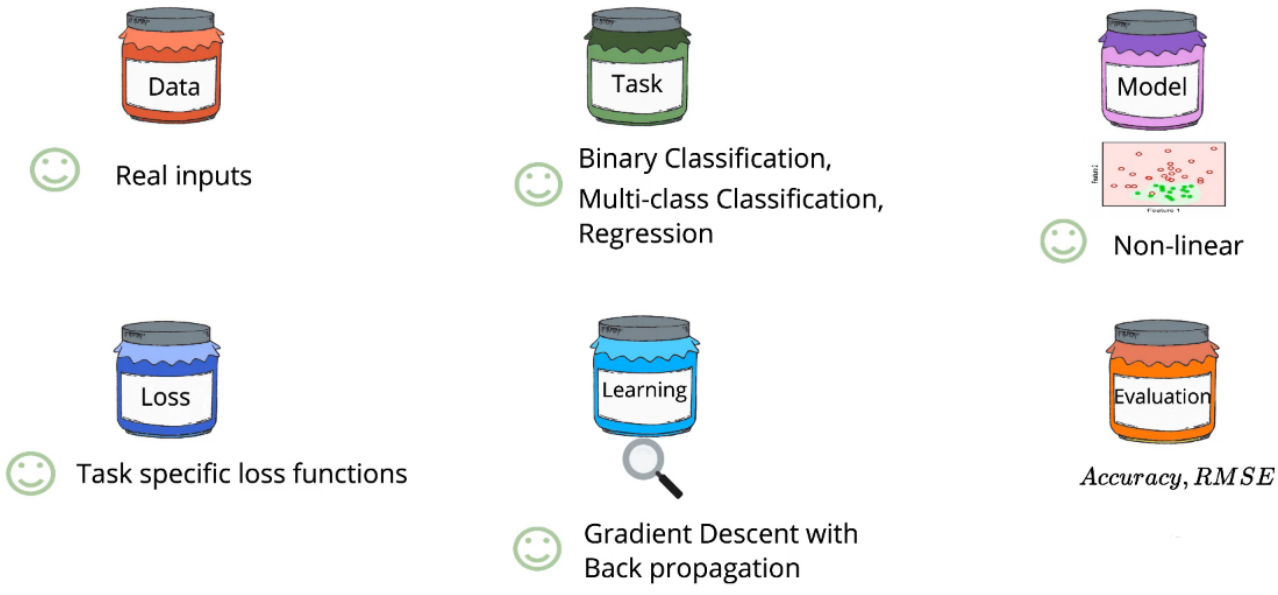
Feedforward Neural Networks — Part 1

P Parveen Khurana Jan 21, 2020 · 16 min read

This article covers the content discussed in the Feedforward Neural Networks module of the Deep Learning course and all the images are taken from the same module.

So far, we have discussed the **MP Neuron, Perceptron, Sigmoid Neuron model** and **none of these models are able to deal with non-linear data**. Then in the last article, we have seen the **UAT which says that a Deep Neural Network can approximate the relationship between the input and the output no matter how complex the relationship between the input and the true output is**.

In this article, we discuss the Feedforward neural network and the situation is going to be like the below with respect to 6 jars of ML.

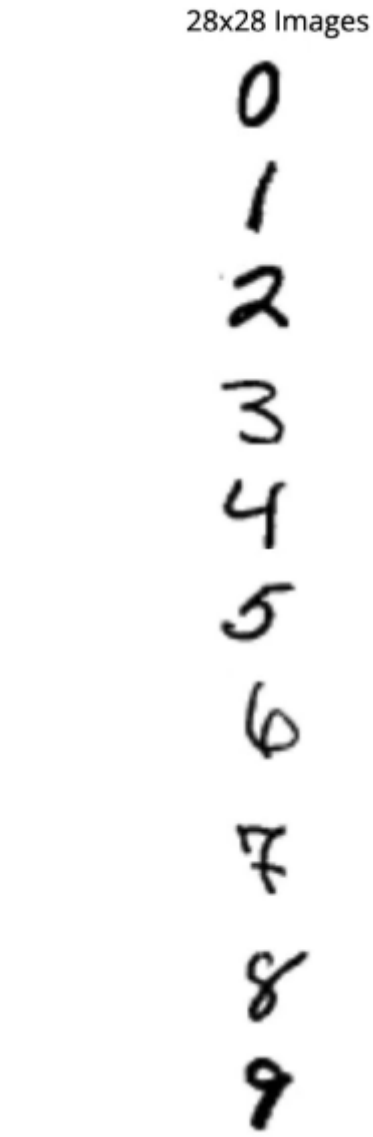


functions and not just the squared error loss.

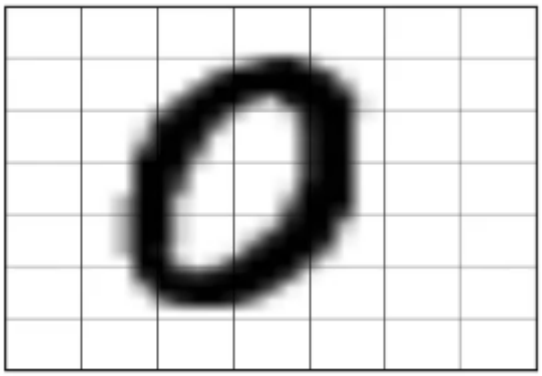
Data and Tasks:

Let's look at what data and tasks DNNs have been used for:

First is the MNIST dataset, the task here is that given an image, we have to identify which of the 10 digits(0 to 9) that it belongs to:



We can think of each pixel as a value here



How can we represent MNIST images as a vector ?

- Using pixel values of each cell
- Matrix having pixel values will be of size 28x28 (As MNIST images are of size 28x28)
- Each pixel value can range from 0 to 255. Standardise pixel values by dividing with 255

255	183	95	8	93	196	253
254	154	37		28	172	254
252	221	...	...	...	...	...
...	...	...	...	...	...	...
...	...	...	...	...	...	...
...	...	...	...	...	198	253
252	250	187	178	195	253	253

We can standardize the data:

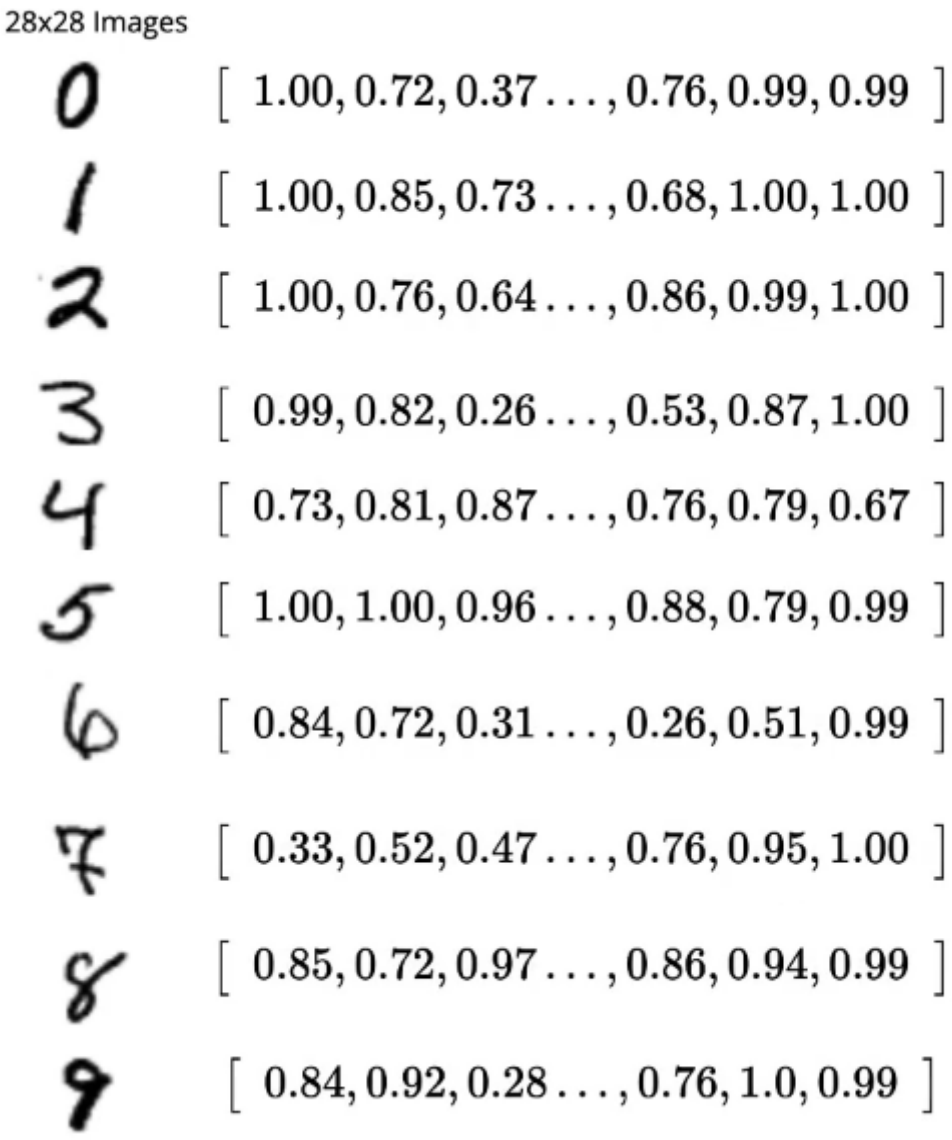
1	0.72	0.37	0.03	0.36	0.77	0.99
1	0.60	0.14		0.11	0.67	1
0.99	0.87	...	...	...	...	...

...	...	...	...	...	0.78	0.99
0.99	0.98	0.73	0.69	0.76	0.99	0.99

- Now, Flatten the matrix to convert into a vector of size 784 (28x28)

We can now flatten this 28 X 28 matrix into a single vector which would be of dimension 784.

Similarly, we can convert all the images to a vector



multi-class classification problem. The class labels we can represent as a one-hot vector or one hot representation:

28x28 Images	Class Label	Class Labels - One hot Representation
 [1.00, 0.72, 0.37 . . . , 0.76, 0.99, 0.99]	0	[1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
 [1.00, 0.85, 0.73 . . . , 0.68, 1.00, 1.00]	1	[0, 1, 0, 0, 0, 0, 0, 0, 0, 0]
 [1.00, 0.76, 0.64 . . . , 0.86, 0.99, 1.00]	2	[0, 0, 1, 0, 0, 0, 0, 0, 0, 0]
 [0.99, 0.82, 0.26 . . . , 0.53, 0.87, 1.00]	3	[0, 0, 0, 1, 0, 0, 0, 0, 0, 0]
 [0.73, 0.81, 0.87 . . . , 0.76, 0.79, 0.67]	4	[0, 0, 0, 0, 1, 0, 0, 0, 0, 0]
 [1.00, 1.00, 0.96 . . . , 0.88, 0.79, 0.99]	5	[0, 0, 0, 0, 0, 1, 0, 0, 0, 0]
 [0.84, 0.72, 0.31 . . . , 0.26, 0.51, 0.99]	6	[0, 0, 0, 0, 0, 0, 1, 0, 0, 0]
 [0.33, 0.52, 0.47 . . . , 0.76, 0.95, 1.00]	7	[0, 0, 0, 0, 0, 0, 0, 1, 0, 0]
 [0.85, 0.72, 0.97 . . . , 0.86, 0.94, 0.99]	8	[0, 0, 0, 0, 0, 0, 0, 0, 1, 0]
 [0.84, 0.92, 0.28 . . . , 0.76, 1.00, 0.99]	9	[0, 0, 0, 0, 0, 0, 0, 0, 0, 1]

So, we can say that the output is a random variable which can take on 10 values from 0 to 9 and in this particular case since we know that the input image is a 0, all the probability mass is focused on the first label or the first event which is the event that the image takes on a value 0. And similarly, for all the other labels we could represent it as a one-hot vector so that later on we can say that this is the true distribution and at some point, we would predict the distribution and can then compare it with the true distribution.

Another category of problems for which the DNNs have been used are:

Age	Albumin	T_Bilirubin		D
65	3.3	0.7		0
62	3.2	10.9		0
20	4	1.1		1
84	3.2	0.7		1
⋮	⋮	⋮	⋮	⋮

where we are given data about the patients and based on that we have to decide whether this patient has some liver disease or not. So, this is again a classification problem(binary).

The third type of problem is regression where we have been given some data about a locality and we want to predict the value of a house there.

Boston Housing* - Predict Housing Values in Suburbs of Boston

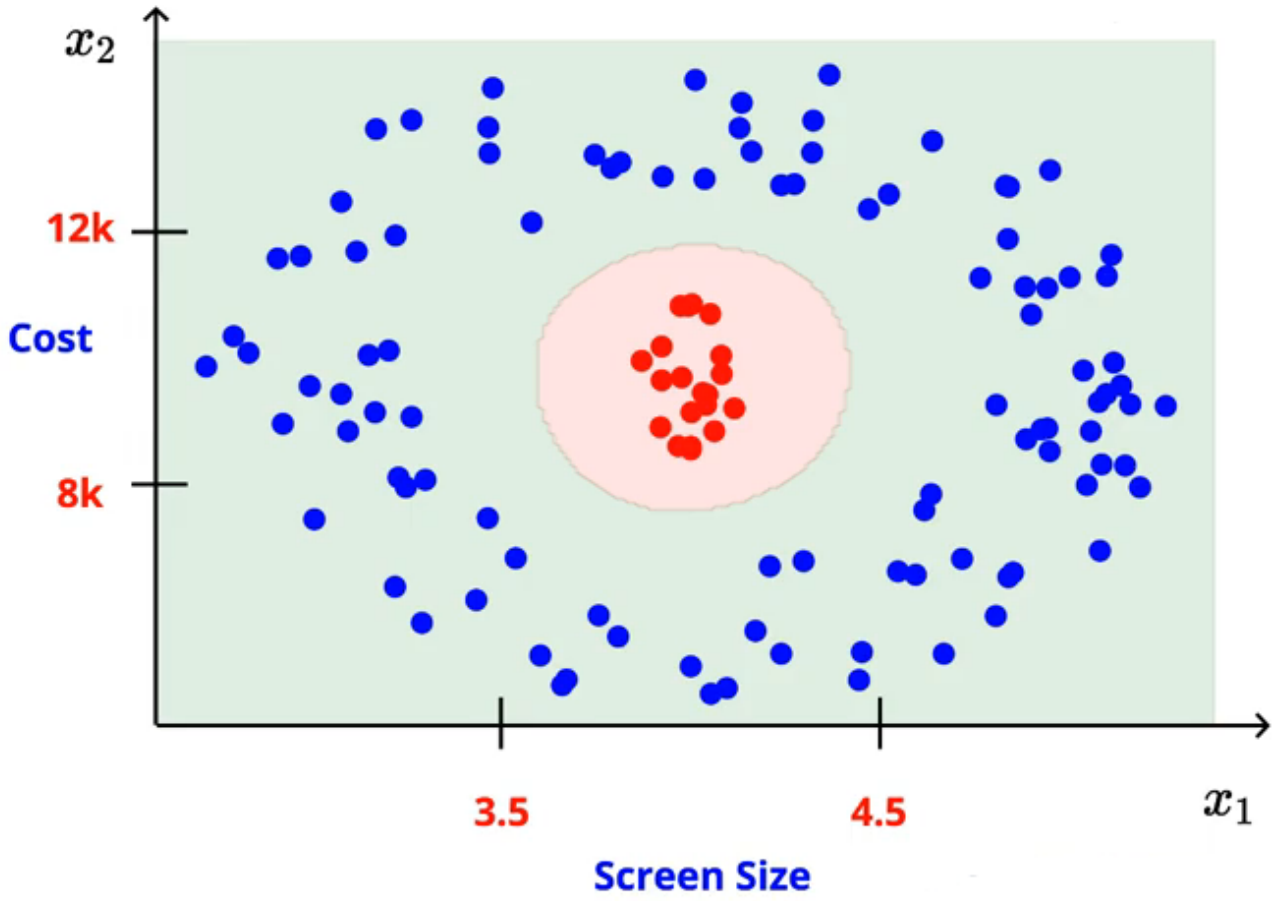
Crime	Avg No of rooms	Age		House Value
0.00632	6.575	65.2		24
0.02731	6.421	78.9		21.6
0.3237	6.998	45.8		33.4
0.6905	7.147	54.2		36.2
⋮	⋮	⋮	⋮	⋮

So, DNNs have been used for many tasks of which we will discuss in detail the below ones:

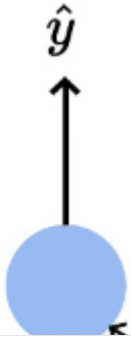
i.) Multi-class classification

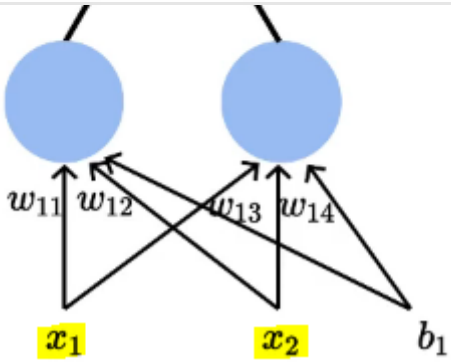
ii.) Binary classification

Let's say we have the below data for a case where we want to find a model to approximate the relationship between the Screen Size and Cost.



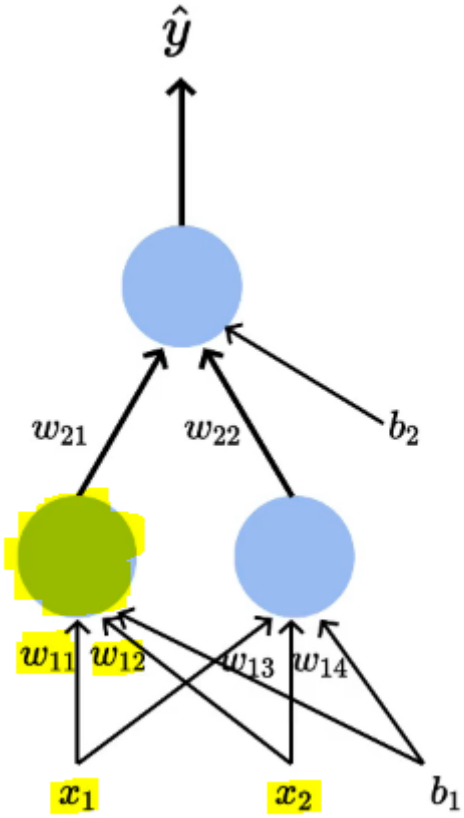
And we want our model to approximate the relation between the input and the output. Clearly the above data is not linearly separable and as discussed in the last article, a sigmoid neuron would not be able to fit this data well no matter how we adjust its parameters. So, a single sigmoid neuron would not be able to help us in this situation. From here, we go to a simple network of neurons(a total of 3 sigmoid neurons, 2 in the first layer and 1 in the second layer)





x1 and **x2**(highlighted in the above image) refer to the input data, **x1** corresponds to screen size and **x2** refers to the cost.

Now the **first neuron** is connected to the inputs **x1** and **x2** via the weights **w11** and **w12**(below image). This first neuron we are referring to as **h1**.

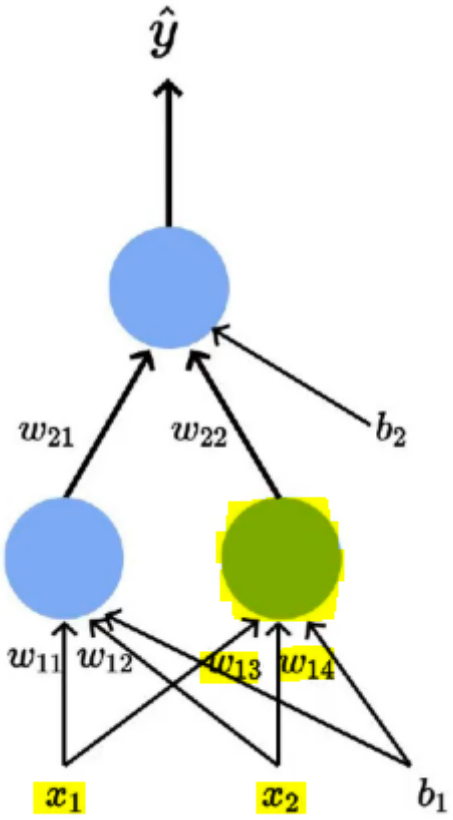


So, we can write that **h1** is a function of **x1** and **x2** and this function has the parameters **w11** and **w12** and **the bias term**:

$$h_1 = f_1(x_1, x_2)$$

$$h_1 = \frac{1}{1+e^{-(w_{11}*x_1+w_{12}*x_2+b_1)}}$$

Now we have this **second neuron** which is taking **x1** and **x2** as the inputs having corresponding weights as **w13** and **w14**(below image). Let's say this is **h2**:

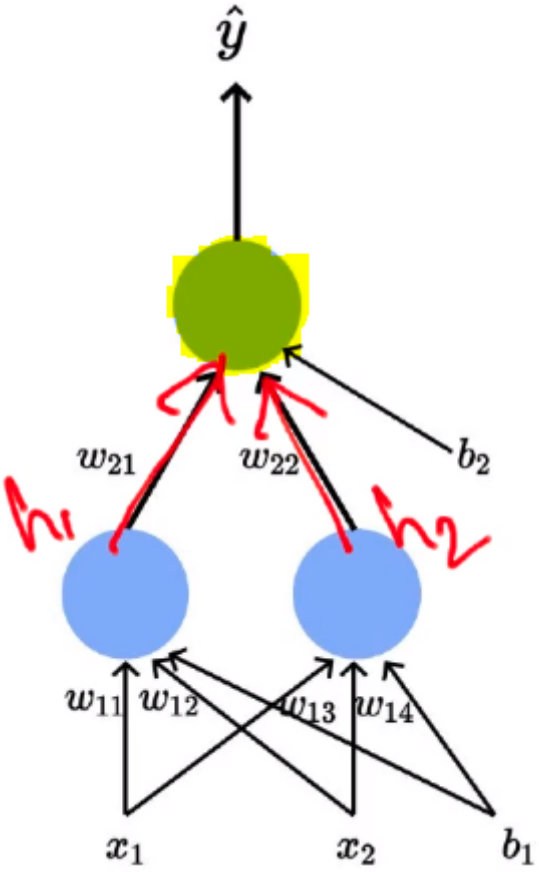


So, **h2** is also a function of **x1** and **x2**, and this function has the parameters **w13**, **w14** and the **bias term**:

$$h_2 = f_2(x_1, x_2)$$

$$h_2 = \frac{1}{1+e^{-(w_{13}*x_1+w_{14}*x_2+b_1)}}$$

Now we have **another neuron in Layer 2**(yellow highlighted in the below image) which takes **h1** and **h2** as the **input**



Now we have the **final output which is a function of h1 and h2**.

$$\hat{y} = g(h_1, h_2)$$

And the **parameters of this function are w21, w22, and b2**.

$$\hat{y} = \frac{1}{1+e^{-(w_{21}*h_1+w_{22}*h_2+b_2)}}$$

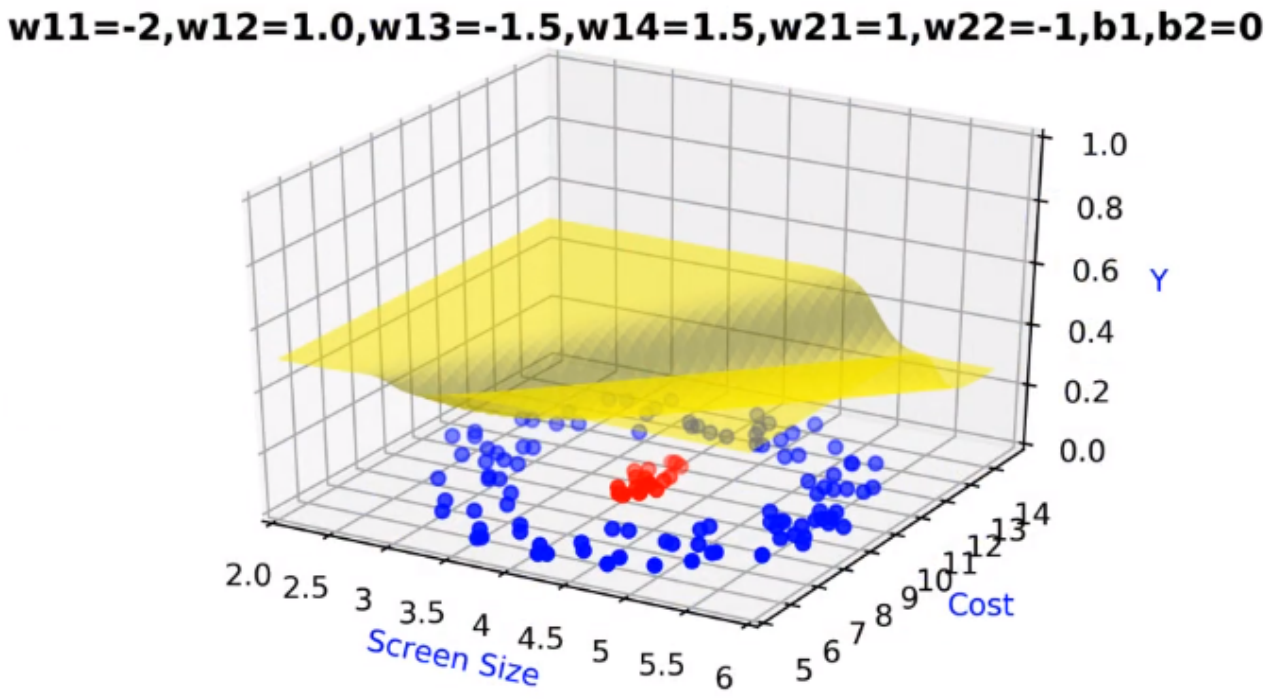
function of x1 and x2. Therefore, we can write the above equation as:

$$\hat{y} = \frac{1}{1+e^{-(w_{21}*h_1+w_{22}*h_2+b_2)}}$$

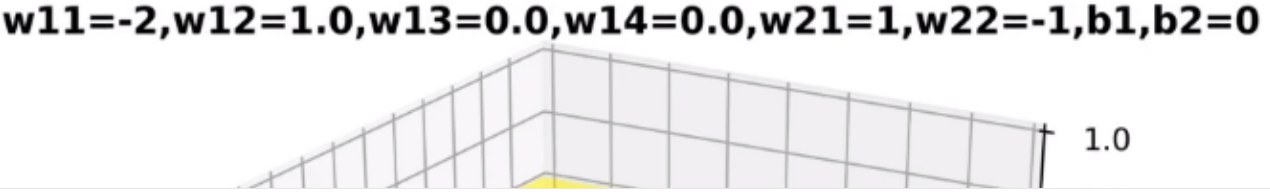
$$= \frac{1}{1+e^{-(w_{21}*(\frac{1}{1+e^{-(w_{11}*x_1+w_{12}*x_2+b_1)}})+w_{22}*(\frac{1}{1+e^{-(w_{13}*x_1+w_{14}*x_2+b_1)}})+b_2)}}$$

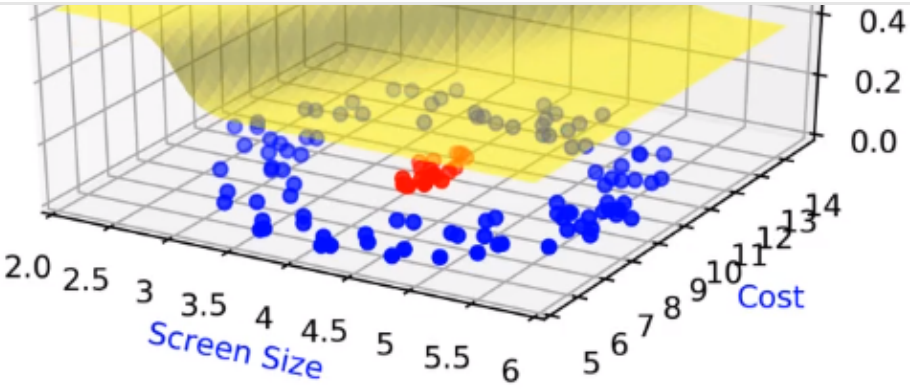
So, we have this final output as a very complex function of the inputs starting with very basic building blocks which is a sigmoid neuron.

We could have a different value of the bias term for each of the neurons. So, considering this, we have a total of 9 parameters(6 weights and 3 bias) for the above case. And we could all of the 9 parameters and the net effect of this looks like:

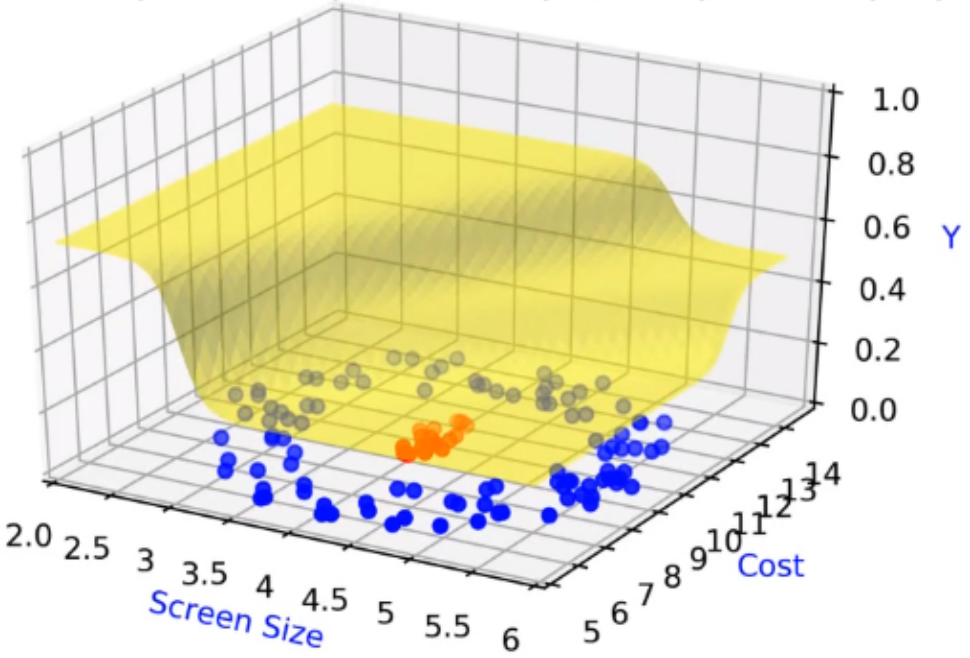


Adjusting the parameters:

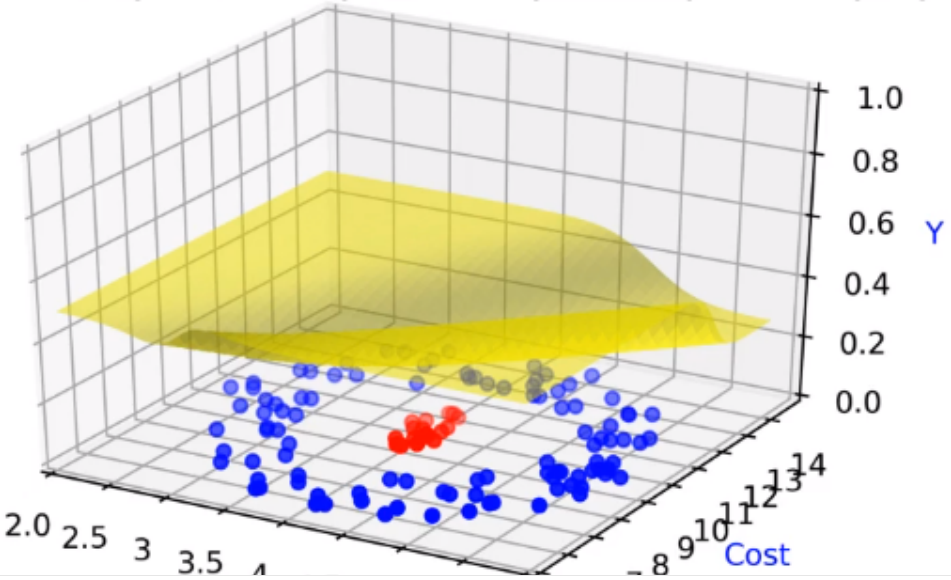


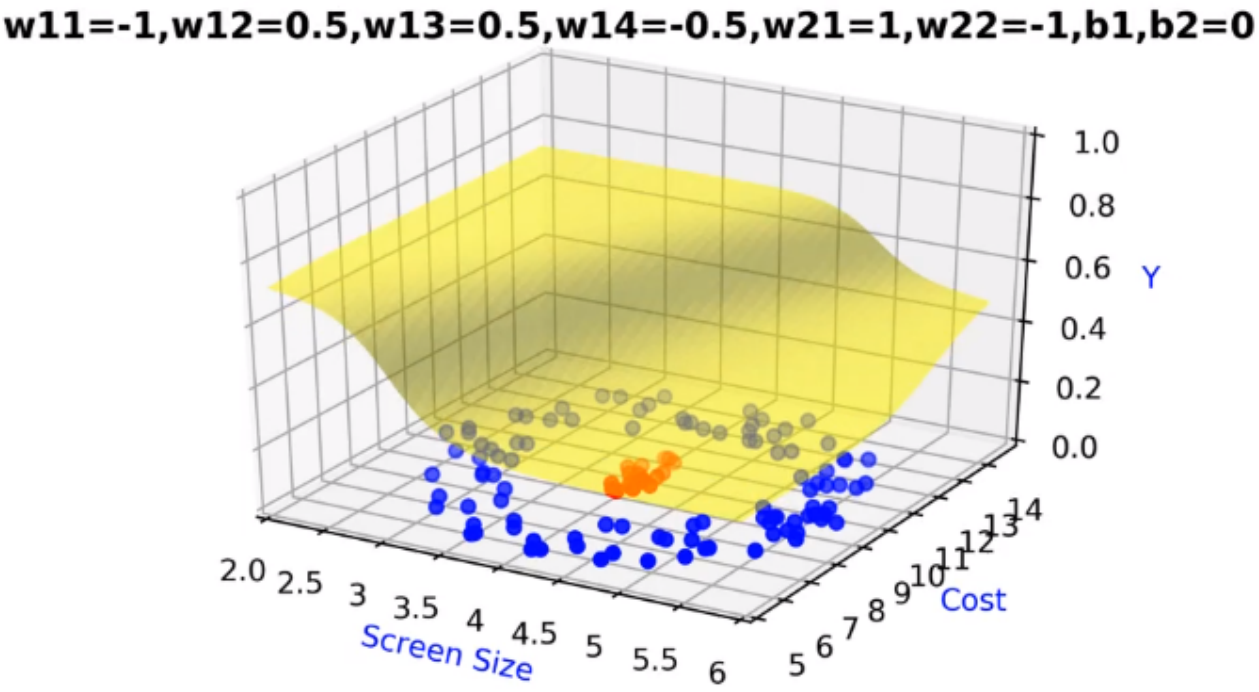


w11=-2,w12=1.0,w13=2.0,w14=-2.0,w21=1,w22=-1,b1,b2=0

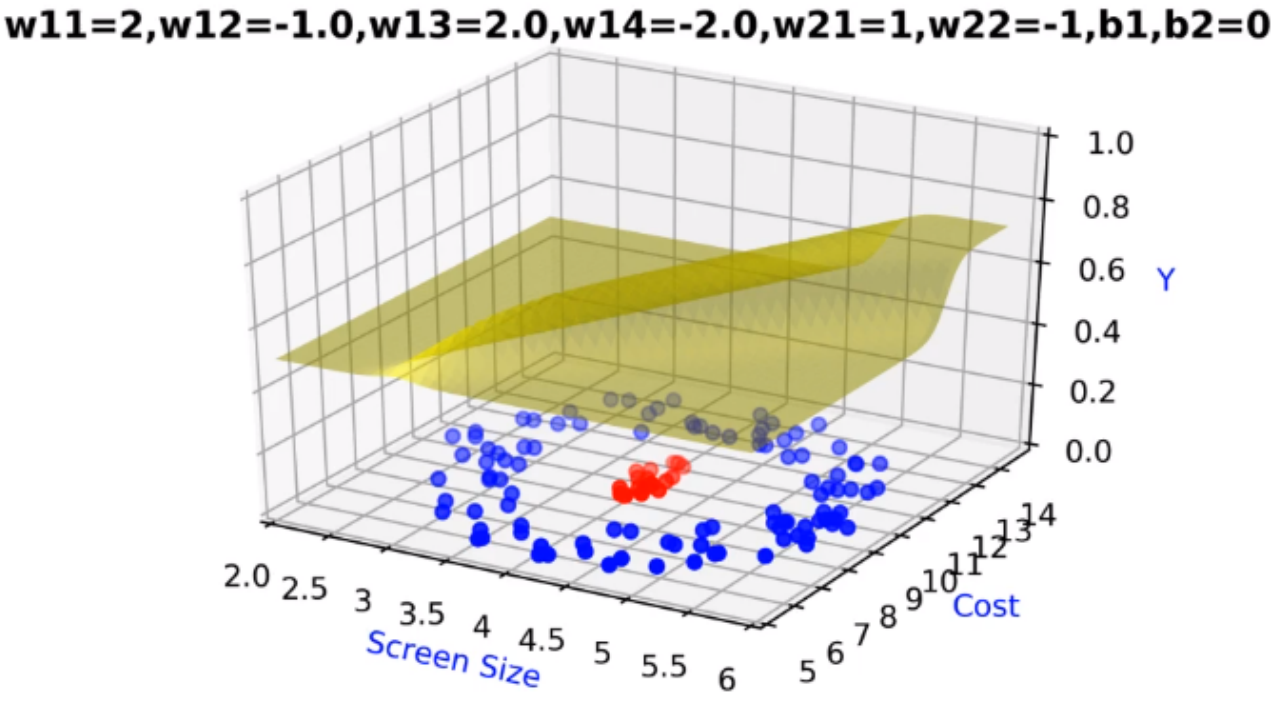


w11=-1,w12=0.5,w13=-2.0,w14=2.0,w21=1,w22=-1,b1,b2=0



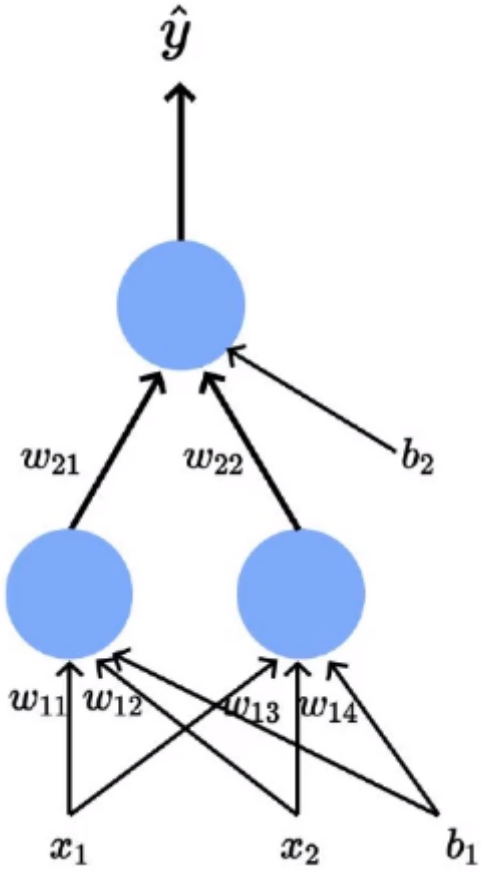


We can keep changing the parameters and try out different configurations and interesting thing to note is that we would end up with some kind of surface which exactly meets the training data



So, in a very simple neural network having only 2 layers and 3 neurons(1 output layer containing 1 neuron, 1 intermediate layer having 2 neurons), we are able to fit the

For the time being, we would focus on the UAT which says that we can approximate the relationship between the input and the output using a deep neural network no matter how complex the true relationship is. And this thing whether to use two layers or three layers or more and the numbers of neurons in each layer is known as **hyper-parameters** which will discuss in a different article.



A generic deep neural network:

So, we are given input and in general, this **input** could be **n-dimensional**, for simplicity we have taken 3 inputs in this case, then we have a certain number of layers of neurons for example, in this case, we have 2 intermediate layers and then the output layer:

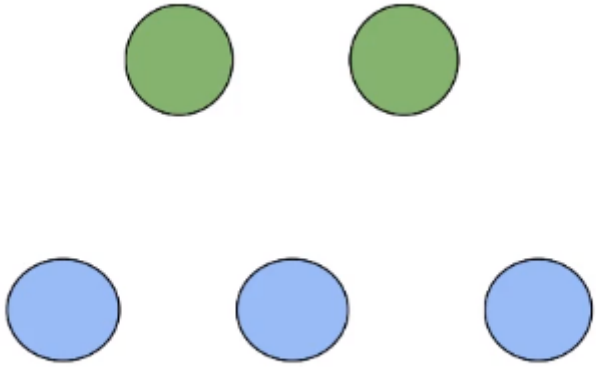
$x_1 \quad x_2 \quad x_3$

The **very first layer is known as the input layer** which is just **x1, x2, x3 in the above image** and the **last layer is known as the output layer** which contains two green circles in the above image. **All other layers between the input and the output layer are known as the intermediate/hidden layers.**

For the time being, we would not worry about how many layers we have and how many neurons we have in each layer.

Let's say **in general** we have **L intermediate layers** and **one input and one output layer**. Each of the L intermediate layers can have a different number of neurons, **the first intermediate layer has say 'm1' neurons**, **the second layer has 'm2' neurons** and **so on and the last intermediate layer has say 'mL' neurons.**

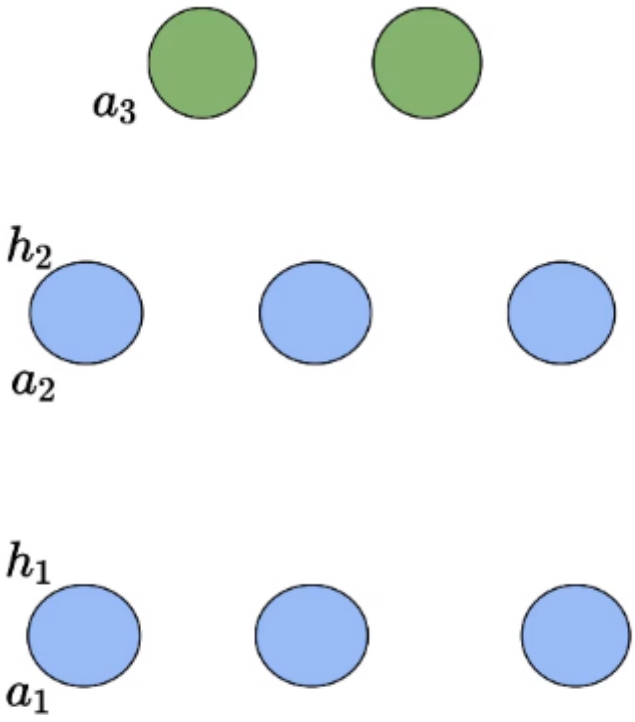
For the below case, we have the same no. of neurons in each of the intermediate layers



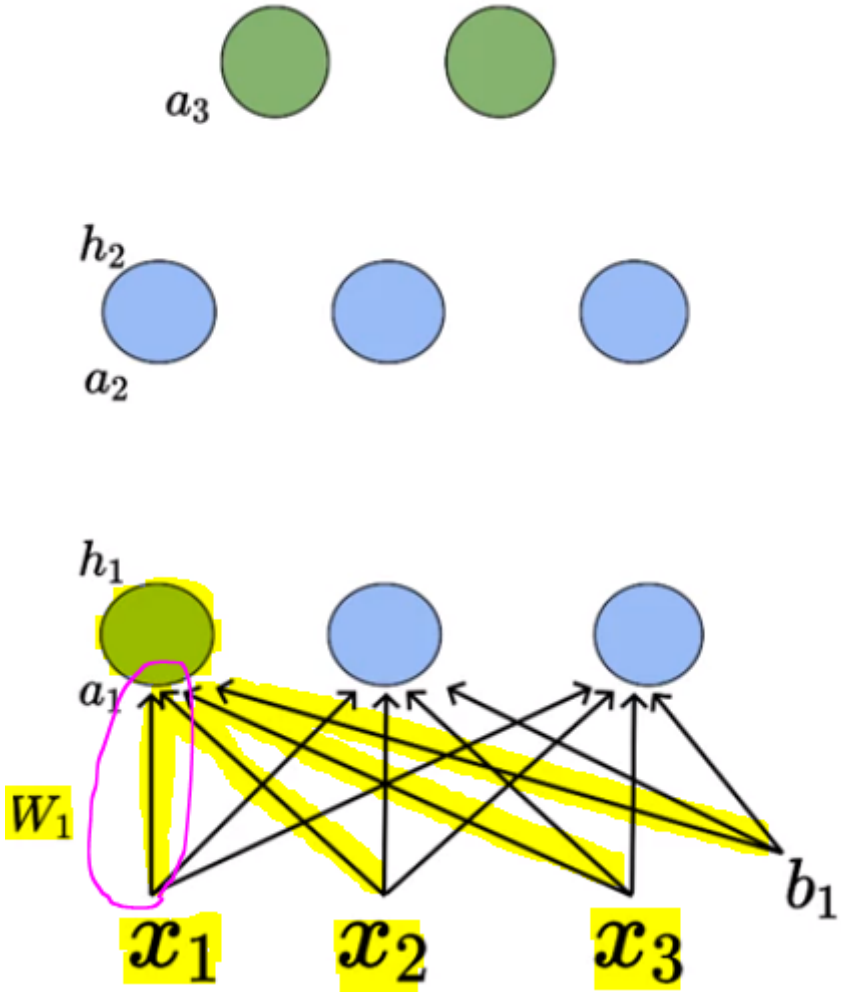
$x_1 \quad x_2 \quad x_3$

Now **for each of the neurons**, **we have two things** happening, **one is the pre-activation which we would denote by 'a' and then the activation which we would call 'h'.**

Just like in the Perceptron case, the function first aggregates the inputs and based on that output some value, so **this aggregation of the inputs is pre-activation and whatever it does after that is activation. In the case of Sigmoid function, this pre-activation is summation/aggregation and the activation is passing this aggregate value through a Sigmoid function.**



Now each of the neurons in the first intermediate layer is connected to all the neurons/inputs in the input layer



Let's say this is Layer 1, so we are going to call all the weights of Layer 1 starting with suffix 1, so the first weight(circled in pink in the above image) is the weight connecting the first neuron in the hidden layer to the first neuron in the input layer, so we could refer to this weight as:

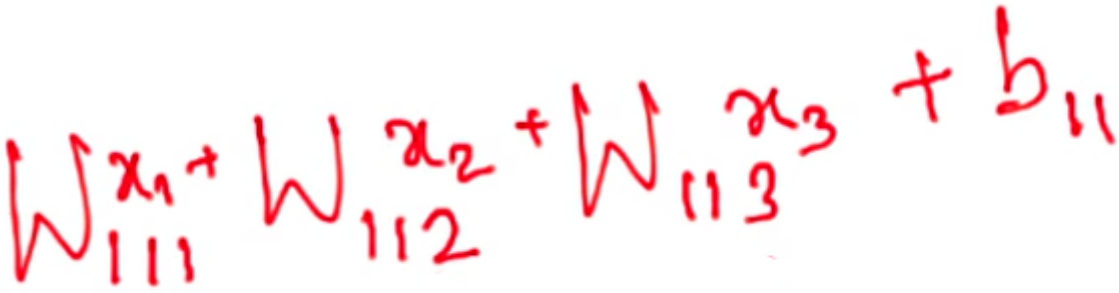
The next weight we could write as:

First 1 denotes the layer number and within that its the first neuron in the intermediate layer, so the second 1 in the denotion is for the neuron number to say and after that 2 denotes the second input from the input layer.

Similarly, we have

The pre-activation of this neuron can be denoted as the below since its the first neuron in the first intermediate layer

The value of the pre-activation would be:



b11 denotes the bias for the first neuron in the first intermediate layer.

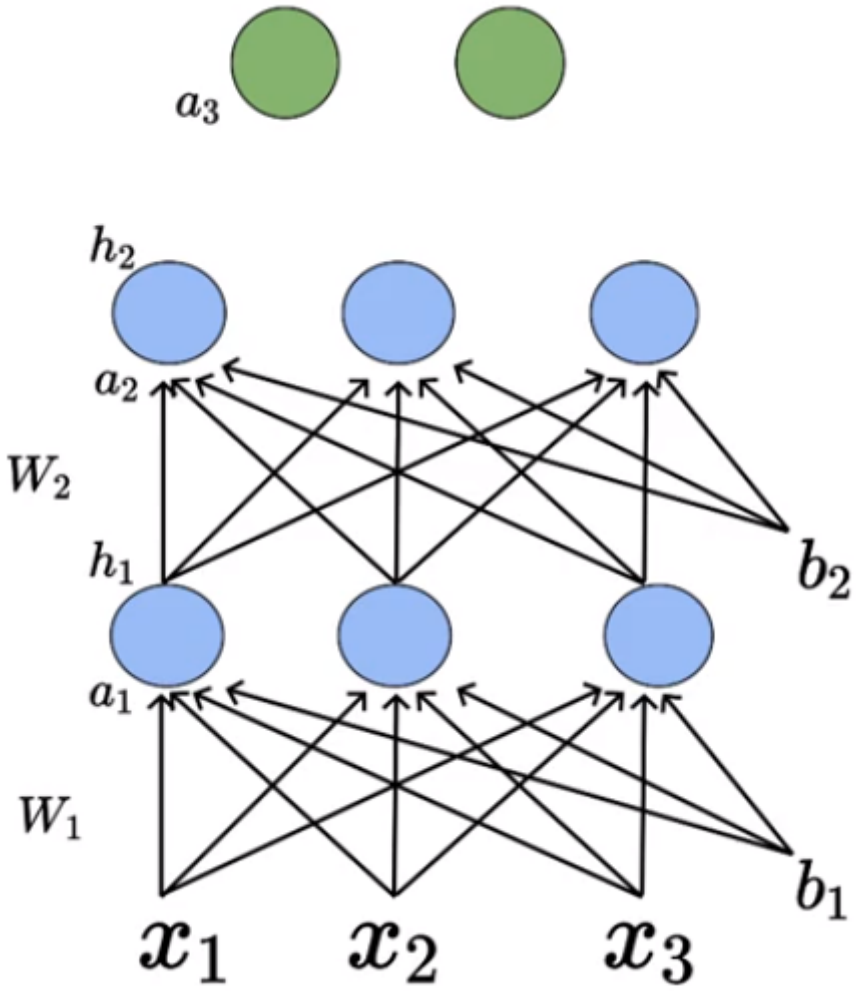
$$h_{11} = \frac{1}{1 + e^{-a_{11}}}$$

Now, let's suppose we have 100 neurons in the input layer and 10 neurons in the first intermediate layer, so now each of the input neurons are going to be connected to each of the hidden neurons by some weight. So, in total, we need 100 X 10 weights. In other words, we can say that W1 is 100 X 10 matrix. It has all the weights related to the first layer.

$$W_1 \in \mathbb{R}^{100 \times 10}$$

Similarly, we would have one bias term for every neuron in the hidden layer so that means we can think of this bias as the 10-dimensional vector

$$b \in \mathbb{R}^{10}$$

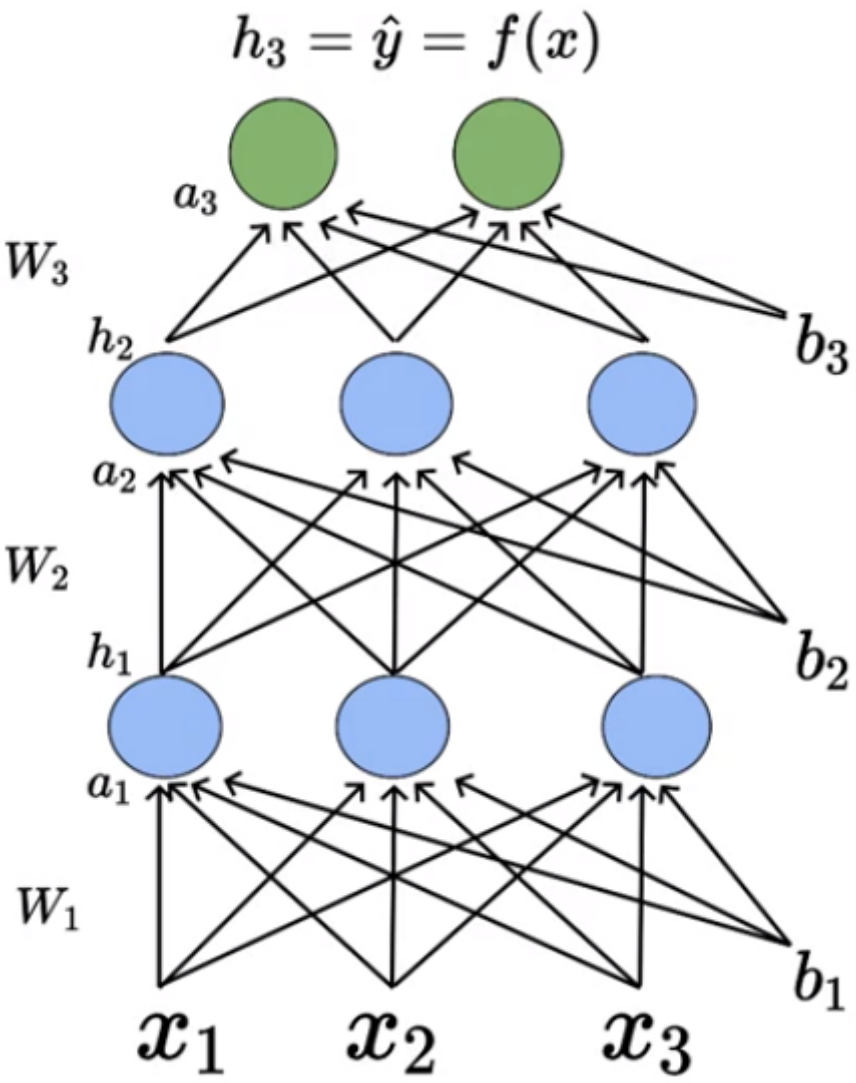


Each of the m1 neurons in the first intermediate layer would be connected to each of the m2 neurons in the second intermediate layer. So, W2 we can think of as 10 X 10 matrix(taking that the first and the second intermediate layer contains 10 neurons). And similarly, there would be one bias for each of the neuron in this layer, so b2 would be a 10-dimensional vector

W2 dimensions

b_2 같다

And these connections go all the way up to the end:



- The pre-activation at layer 'i' is given by

$$a_i(x) = W_i h_{i-1}(x) + b_i$$

- The activation at layer 'i' is given by

$$h_i(x) = g(a_i(x))$$

where 'g' is called as the activation function

$$f(x) = h_L = O(a_L)$$

where 'O' is called as the output activation function

In the above image:

i.) pre-activation is shown as a function of x as all the pre-activation or activation for any of the layers depends directly or indirectly on the input x(input layer).

ii.) The final output layer is denoted by L.

Understanding the computations in a deep neural network:

Let's assume that we have 100 neurons in the input layers and 10 neurons in the first intermediate layer, then W1 is going to have a total of 10 X 100 weights and we can write it as:

$$W_1 = \begin{bmatrix} w_{1\,1\,1} & w_{1\,1\,2} & . & . & . & w_{1\,1\,99} & w_{1\,1\,100} \\ w_{1\,2\,1} & w_{1\,2\,2} & . & . & . & w_{1\,2\,99} & w_{1\,2\,100} \\ . & . & . & . & . & . & . \\ . & . & . & . & . & . & . \\ w_{1\,10\,1} & w_{1\,10\,2} & . & . & . & w_{1\,10\,99} & w_{1\,10\,100} \end{bmatrix}$$

The first row in the above matrix represents all the weights which connects all the 100 inputs to the first neuron in the first hidden layer.

The first index in the weight represents the layer number, the second index represents the neuron number to say in the next layer and the third index represents the neuron in the current layer or the input neuron in the above case.

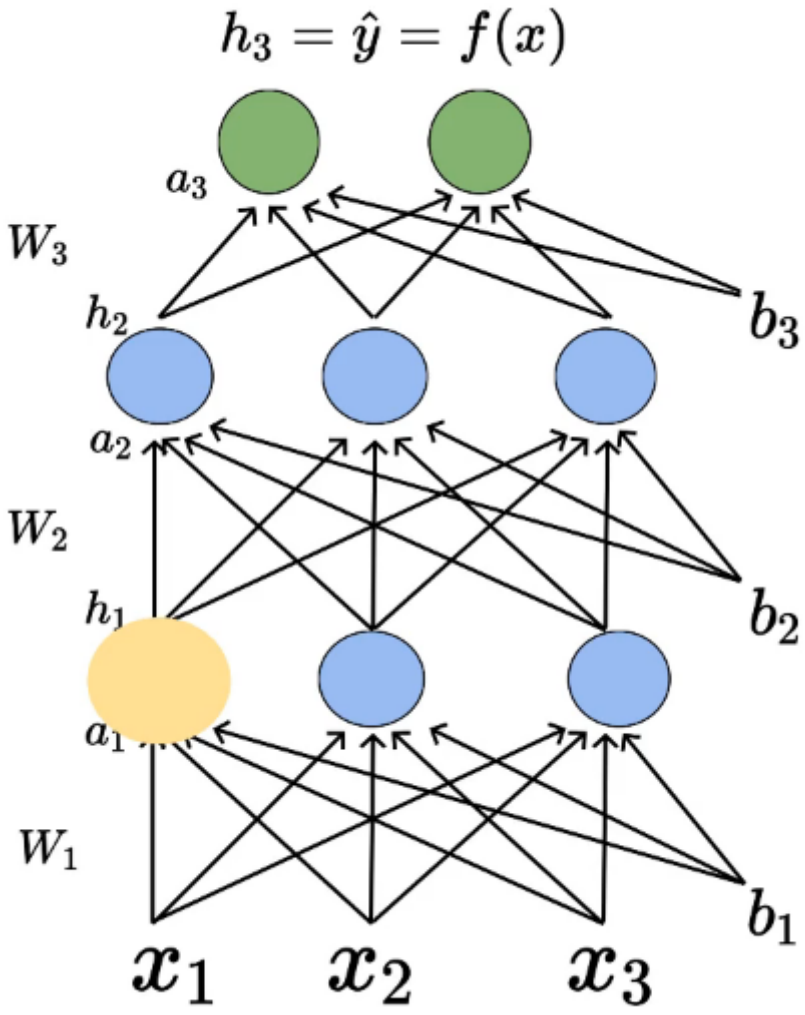
Our input layer consists of 100 neurons and we can think of it as a 100 X 1 vector.

$$\begin{bmatrix} w_{1\,1\,1} & w_{1\,1\,2} & . & . & . & w_{1\,1\,99} & w_{1\,1\,100} \end{bmatrix} \qquad \begin{bmatrix} x_1 \end{bmatrix}$$

$$\begin{bmatrix} w_{1\,10\,1} & w_{1\,10\,2} & . & . & . & w_{1\,10\,99} & w_{1\,10\,100} \end{bmatrix} \quad \begin{bmatrix} x_{100} \end{bmatrix}$$

$\in \mathbb{R}^{10 \times 100}$ $\mathbb{R}^{100 \times 1}$

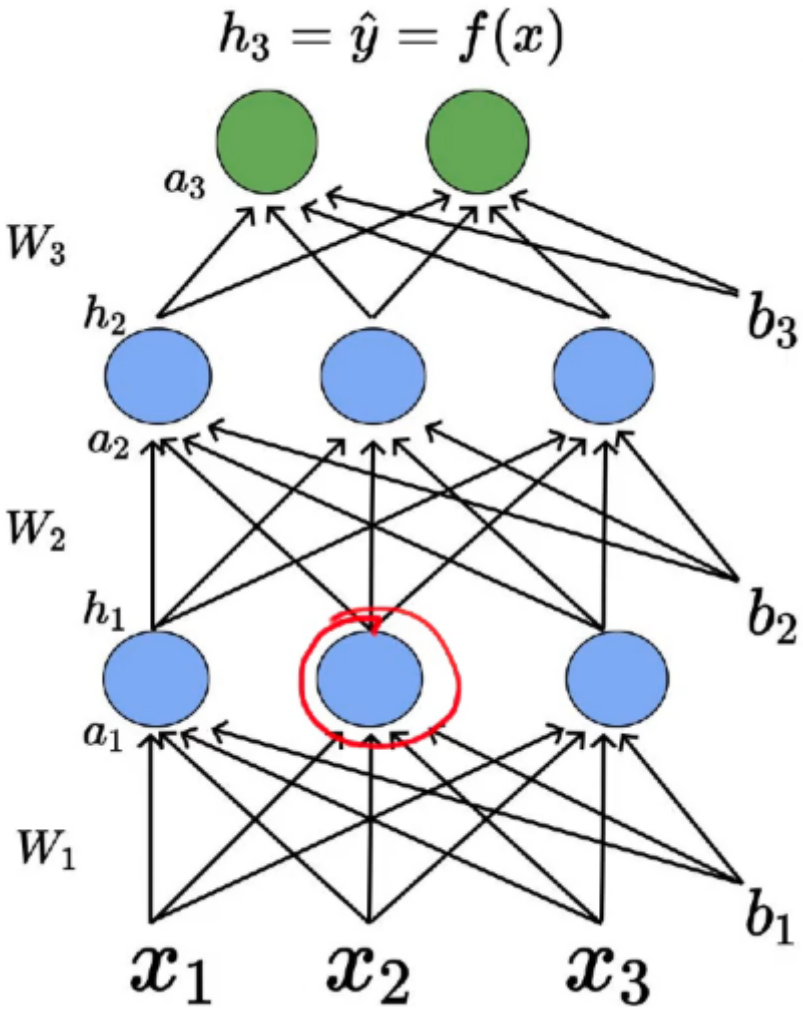
Let's see how to compute a11(pre-activation for the yellow neuron in the below image):



$$W_1 = \begin{bmatrix} w_{1\,1\,1} & w_{1\,1\,2} & . & . & . & w_{1\,1\,99} & w_{1\,1\,100} \\ w_{1\,2\,1} & w_{1\,2\,2} & . & . & . & w_{1\,2\,99} & w_{1\,2\,100} \\ . & . & . & . & . & . & . \\ . & . & . & . & . & . & . \\ w_{1\,10\,1} & w_{1\,10\,2} & . & . & . & w_{1\,10\,99} & w_{1\,10\,100} \end{bmatrix} \quad X = \begin{bmatrix} x_1 \\ x_2 \\ . \\ . \\ x_{100} \end{bmatrix}$$

It would be the weighted sum of all the inputs plus the bias term related to the first neuron in the hidden layer. As is clear from the above equation, this pre-activation value for the first neuron in the first hidden layer is equal to the dot product between the first row from the weights matrix with the input neuron vector plus the bias term.

Similarly, for the second neuron in the first hidden layer can be computed as:



$$W_1 = \begin{bmatrix} w_{1\,1\,1} & w_{1\,1\,2} & . & . & . & w_{1\,1\,99} & w_{1\,1\,100} \\ w_{1\,2\,1} & w_{1\,2\,2} & . & . & . & w_{1\,2\,99} & w_{1\,2\,100} \\ . & . & . & . & . & . & . \\ . & . & . & . & . & . & . \\ w_{1\,10\,1} & w_{1\,10\,2} & . & . & . & w_{1\,10\,99} & w_{1\,10\,100} \end{bmatrix} \quad X = \begin{bmatrix} x_1 \\ x_2 \\ . \\ . \\ x_{100} \end{bmatrix}$$

$$a_{1\,1} = w_{1\,1\,1} * x_1 + w_{1\,1\,2} * x_2 + w_{1\,1\,3} * x_3 + ... + w_{1\,1\,100} * x_{100} + b_{11}$$

So, it is the dot product between the second row of the weight matrix and this column matrix of the data plus the bias term b12(1 represents the layer number and 2 represents the neuron number).

Similarly, we can compute the pre-activation value for all the 10 neurons in the first hidden layer.

$$W_1 = \begin{bmatrix} w_{1\,1\,1} & w_{1\,1\,2} & . & . & . & w_{1\,1\,99} & w_{1\,1\,100} \\ w_{1\,2\,1} & w_{1\,2\,2} & . & . & . & w_{1\,2\,99} & w_{1\,2\,100} \\ . & . & . & . & . & . & . \\ . & . & . & . & . & . & . \\ w_{1\,10\,1} & w_{1\,10\,2} & . & . & . & w_{1\,10\,99} & w_{1\,10\,100} \end{bmatrix} \quad X = \begin{bmatrix} x_1 \\ x_2 \\ . \\ . \\ x_{100} \end{bmatrix}$$

$$a_{1\,1} = w_{1\,1\,1} * x_1 + w_{1\,1\,2} * x_2 + w_{1\,1\,3} * x_3 + ... + w_{1\,1\,100} * x_{100} + b_{11}$$
$$a_{1\,2} = w_{1\,2\,1} * x_1 + w_{1\,2\,2} * x_2 + w_{1\,2\,3} * x_3 + ... + w_{1\,2\,100} * x_{100} + b_{12}$$
$$. \quad . \quad .$$
$$. \quad . \quad .$$
$$a_{1\,10} = w_{1\,10\,1} * x_1 + w_{1\,10\,2} * x_2 + w_{1\,10\,3} * x_3 + ... + w_{1\,10\,100} * x_{100} + b_{1,10}$$

W1 has a dimension of 10 X 100

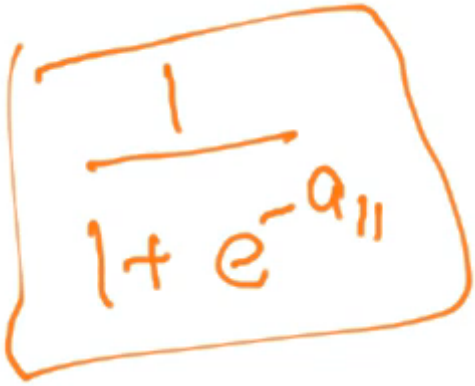
X is a 100 dimensional vector i.e 100 X 1

W1.X would have the dimension as 10 X 1 that is it would be a 10-dimensional vector and all its entries in addition to the terms of the bias vector correspond to the pre-activation value of the 10 neurons in the first hidden layer.

$$a_1 = W_1 * x + b$$

So, we have computed the pre-activation values for all the 10 neurons. Now the activation value would just be the sigmoid applied over the pre-activation value.

For example, **h11** would be:



h12 would be:

In general, we would have:

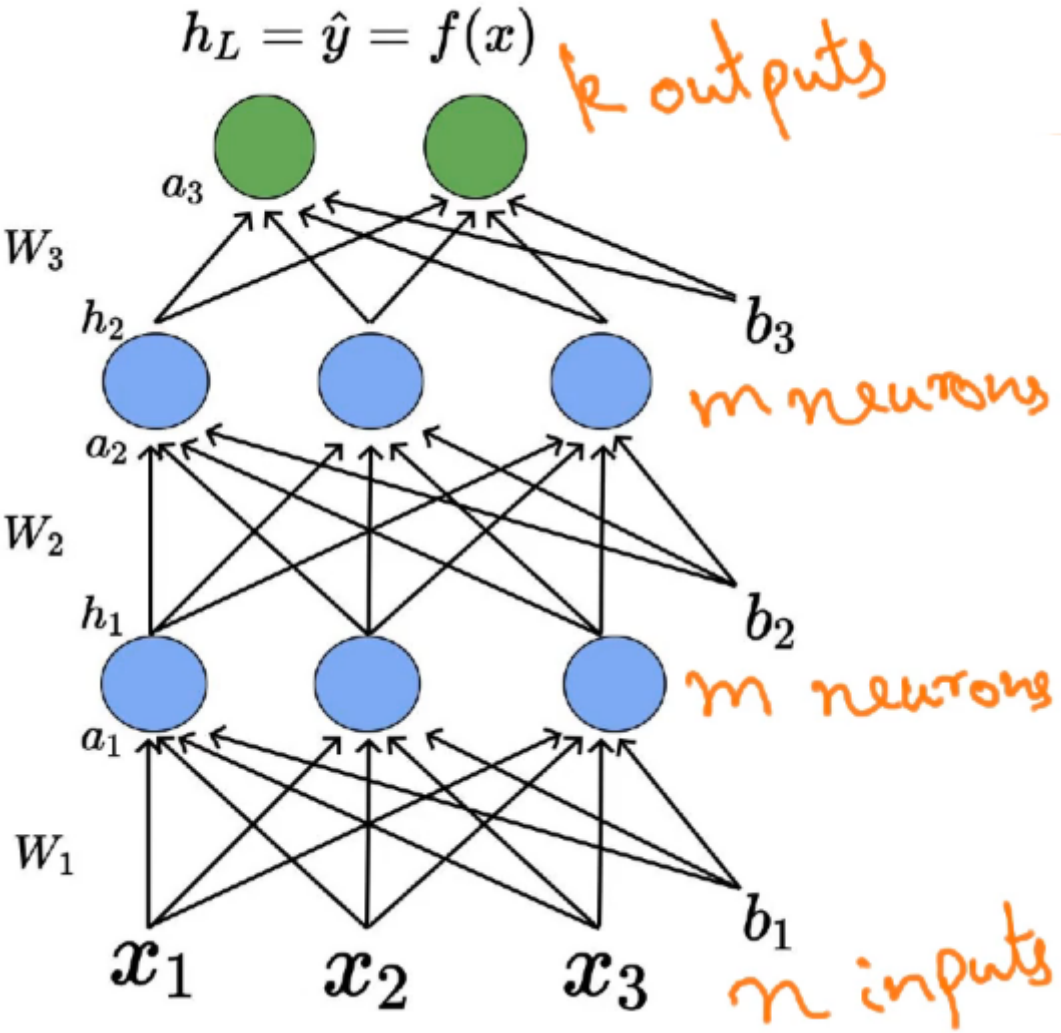
$$h_{11} = g(a_{11}) \qquad h_{12} = g(a_{12}) \quad . \quad . \quad . \quad . \qquad h_{1\,10} = g(a_{1\,10})$$

g represents the sigmoid function

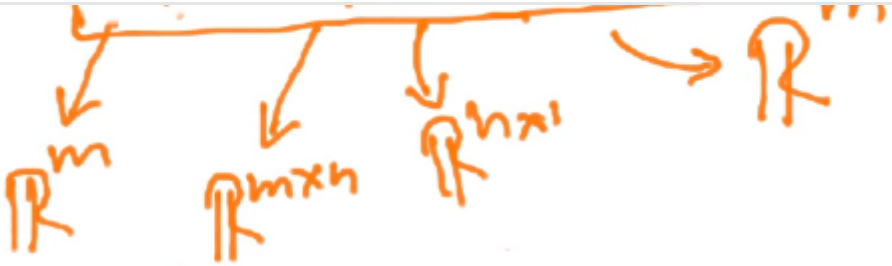
a1 is a 10-dimensional vector and when we apply sigmoid over **a1** that means we apply sigmoid over each and every element of **a1**.

The output layer of a deep neural network:

Let's consider a generic neural network where **we have 'n' neurons in the input layer**, **we have 'L-1' hidden layers** and **each of the hidden layers have the same 'm' neurons**(in practice, it could be different), and **we are trying to produce 'k' outputs**



So, we are can represent all the 'n' input neurons by **X1** vector which would be a 'n' dimensional vector and each of these **'n' input neurons** would connect with each of the **'m' neurons in the first hidden layer**, so the dimensions of the **weight matrix would be 'm X n'** and then we have **bias corresponding to each of the neurons**, so the **bias vector would also be a 'm' dimensional vector**.



a1 would be a **'m' dimensional vector** and **represents the pre-activation value for each of the 'm' neurons in the first hidden layer**.

And then we can apply element-wise sigmoid over **a1** to compute the activation value for each of the '**m**' neurons in the first hidden layer.

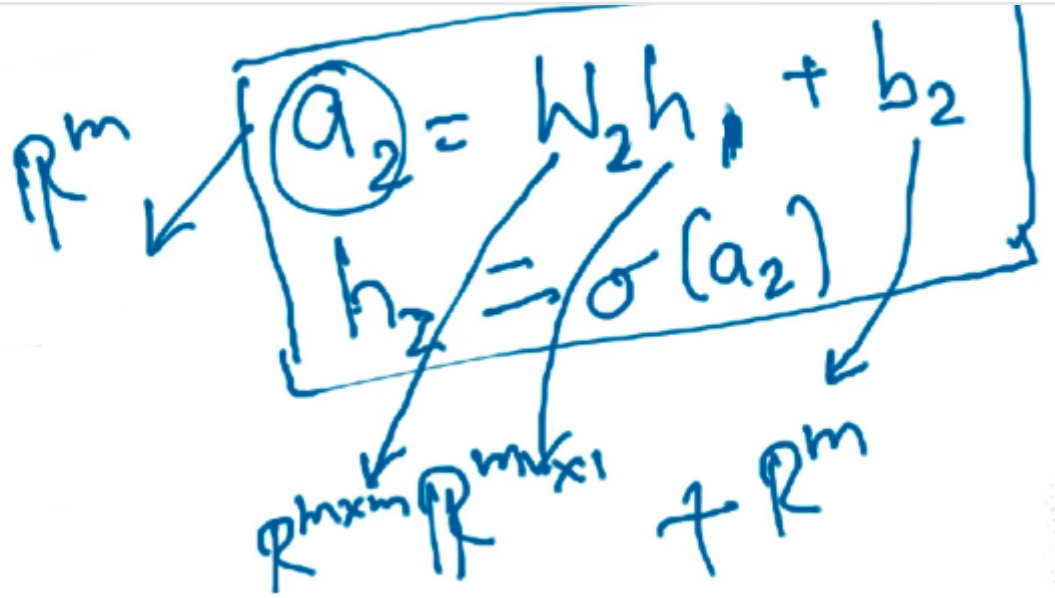
$$h_1 = \sigma(a_1)$$

h1 would also be a 'm' dimensional vector.

And now the same story repeats for the next layer.

$$a_2 = W_2 h_1 + b_2$$
$$h_2 = \sigma(a_2)$$

Each of the '**m**' neurons in the first intermediate layer would be connected to each of the '**m**' neurons in the second intermediate layer, and therefore we have the dimensions of **W2 would be 'm X m'**, **h1**, as discussed above, is going to be a '**m**' dimensional vector and there would be a bias term corresponding to each of the '**m**' neurons in the second intermediate layer so it would be a '**m**' dimensional vector.



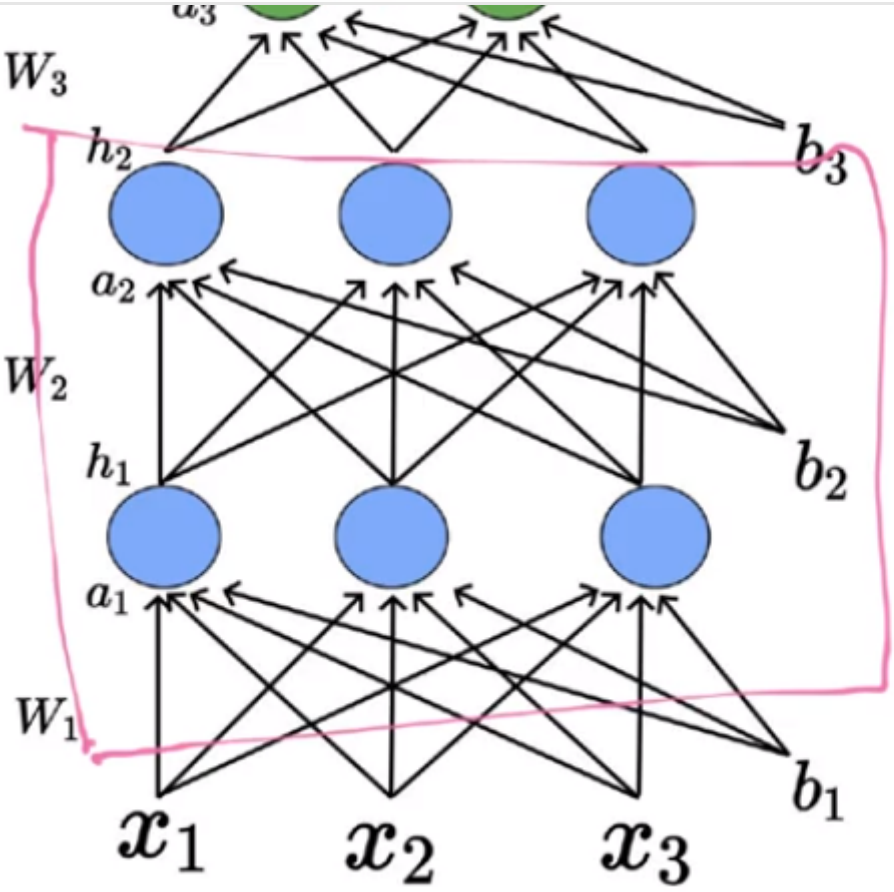
And **once we have the pre-activation values** for each of the '**m**' neurons in the second intermediate layer, we can compute the corresponding sigmoid values for each of the '**m**' neurons and we will get the **h2 vector** which also would be a '**m**' dimensional vector.

Let's suppose at the output we have the case of multi-class classification problem where say there are 4 classes and we want to predict the probability distribution because the true output would be something like the following which is also a probability distribution and the second class is the true class as all the probability mass is focused on this index.

$$[0 \ 1 \ 0 \ 0]$$

And our output is also going to be a probability distribution and we can have some Loss function which computes the difference between these two distributions.

$$h_L = \hat{y} = f(x)$$



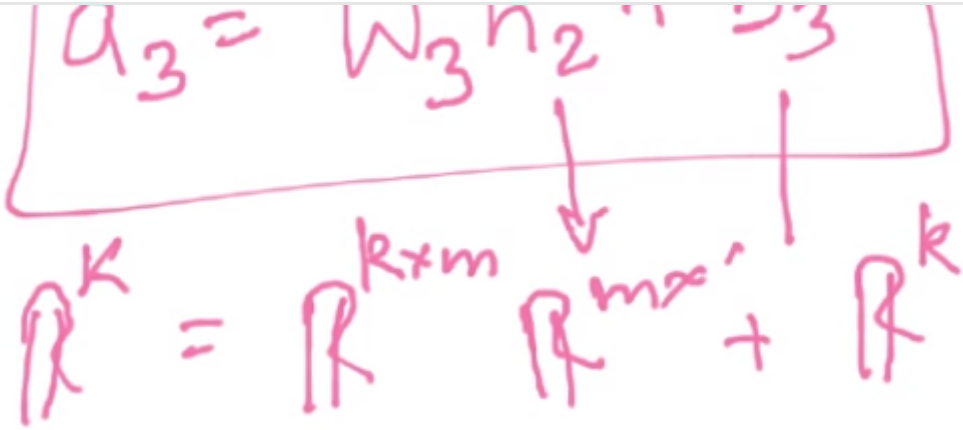
We want the final output to be a probability distribution. The pre-activation values at the output layer would be given as:

$$a_3 = W_3 h_2 + b_3$$

h2 is a **'m' dimensional vector**.

Each neuron in the last intermediate layer('**m**' neurons in this case) would be connected with each of the neurons('**k**' in this case) in the output layer, so the dimensions of **W3 would be 'k X m'**.

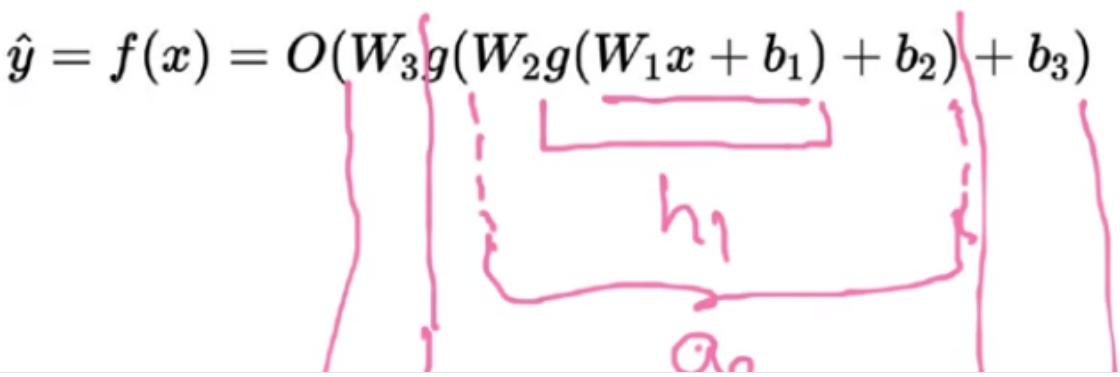
There would be **one bias corresponding to each of the 'k' output neurons**. So, **b3** would be a '**k**' dimensional vector.

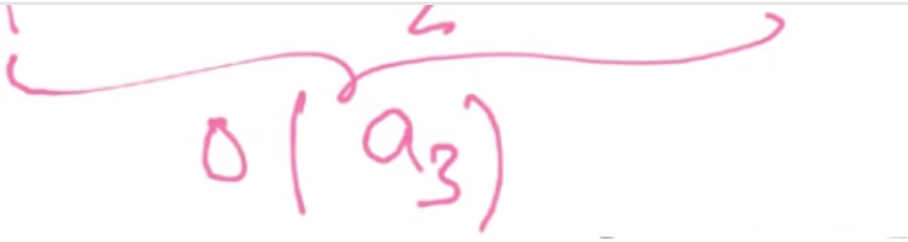


So, **a3** is a '**k**' dimensional vector and from this, we want to predict the final output which is also going to be a '**k**' dimensional vector but we want it to be a probability distribution. So, we can say that the final output is some function of a3:

Before discussing the function O, let's see how we can represent the final output **y_hat** as a function of the inputs:

$$\hat{y} = f(x) = O(W_3 g(W_2 g(W_1 x + b_1) + b_2) + b_3)$$





We can write **y_hat** as a **very composite and complex function of the inputs**, a lot of non-linearity is applied along the way.

How do we decide the output layer depends on the task at hand and is discussed in this article.

Artificial Intelligence Artificial Neural Network Deep Learning Data Science Feedforward Net